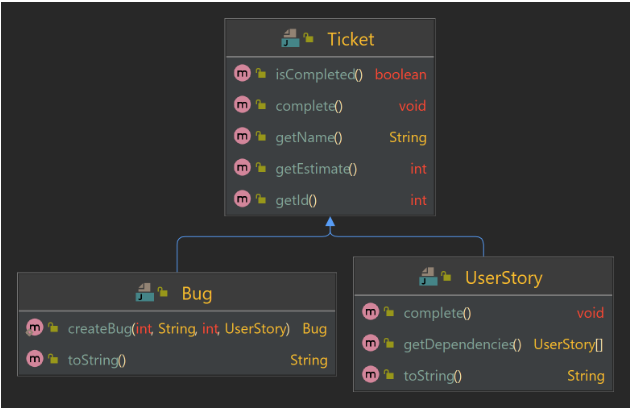
**ЛАБОРАТОРНА РОБОТА №3**

**Базові принципи ООП**

**Мета роботи:** Використовуючи теоретичне підґрунтя про базові принципи ООП та підходи використання в Java виконати дії що будуть вказано в завданні до лабораторної роботи.

**Завдання №1.** У цьому завданні ви формулюватимете бізнес-логіку планування спринту. Спринт - це основна одиниця розробки програмного забезпечення SCRUM. Спринти обмежені за часом, час спринту узгоджується при плануванні. Вам належить організувати планування спринту, що включає завдання, які вам потрібно реалізувати на основі базового класу Ticket. Врахуйте, у цьому випадку спринт приймає лише два підтипи класу Ticket:Bug та UserStory.

Ознайомтеся з діаграмою, яка показує публічний програмний інтерфейс цих класів та їхні відношення:



**Ticket**

Кожен ticket має свій ідентифікатор, ім'я та приблизний час для його виконання. Ці значення надаються через конструктор класу Ticket.

Ticket може вважатися завершеним або незавершеним. Щойно створений ticket вважається незавершеним.

• getId() - повертає ідентифікатор ticket'у.

• getName() - повертає ім'я ticket'у.

• getEstimate() - повертає оцінку ticket'у.

• isCompleted() - повертає true, якщо ticket завершено, інакше — false.

• complete() - переводить ticket у завершений стан.

**UserStory**

UserStory (історія користувача) представляє ticket, який може містити залежності. Залежності — це інші екземпляри UserStory, які мають бути завершені в першу чергу, щоб потім вдалося завершити і залежне від них UserStory. Залежності передаються через конструктор класу UserStory.

• complete() - як і метод Ticket#complete() - переводить ticket у завершений стан. Різниця лише в тому, що історія користувача може не бути завершена, якщо його залежність ще не завершена.

• getDependencies() - повертає копію масиву ticket'ів на спринт, що захищена від змін.

• toString() - повертає рядок, що представляє UserStory, використовуючи його ідентифікатор та ім'я. Приклад: з ідентифікатором =1, ім’я UserStory = "Об'єкт реєстрації користувача", отримаємо "[US 1] Об'єкт реєстрації користувача".

**Bug**

Багом вважається ticket, що відноситься до завершеної UserStory. Баги не існують самі по собі, без відповідного екземпляра UserStory.

• createBug(int id, String name, int estimate, UserStory userStory) - статичний метод створення екземпляра Bug. Повертає null, якщо UserStory має значення null або не завершено. Інакше повертає створений екземпляр Bug.

• toString() - повертає рядкове представлення цього бага, використовуючи ідентифікатор, ім'я та ім'я відповідної UserStory. Приклад: з ідентифікатором = 2, ім'ям бага = "Додати пароль, що повторюється" і ім'ям відповідної UserStory = "Форма реєстрації" в результаті отримаємо "[Bug 2] Форма реєстрації: Додати повторюваний пароль".

**Sprint**

Спринти мають тимчасову ємність та обмеження кількості ticket'ів, які задаються через конструктор. Спринт не може містити тікет з сумарною оцінкою часу виконання, що перевищує тимчасову ємність спринту. Не допускається, щоб спринт містив більше тикетів, ніж визначено обмеженням кількості тикетів для цього спринту. Спринт повинен приймати ticket'и за допомогою методів add\*. Ці методи повертають true, коли вхідний ticket був прийнятий у спринт, і false в іншому випадку.

Спринт не повинен приймати

1. Значення null.

2. ticket'и, які вже завершені.

3. ticket'и, що мають значення оцінки часу виконання, яке, у разі додавання ticket'а, призведе до переповнення тимчасової ємності спринту.

4. Будь-який ticket, якщо досягнуто межі кількості ticket'ів у спринті.

Опис методів спринта

• addUserStory(UserStory userStory) - приймає userStory, якщо вона не дорівнює null, і не завершена. Повертає true, якщо історія користувача прийнята, у противному випадку false.

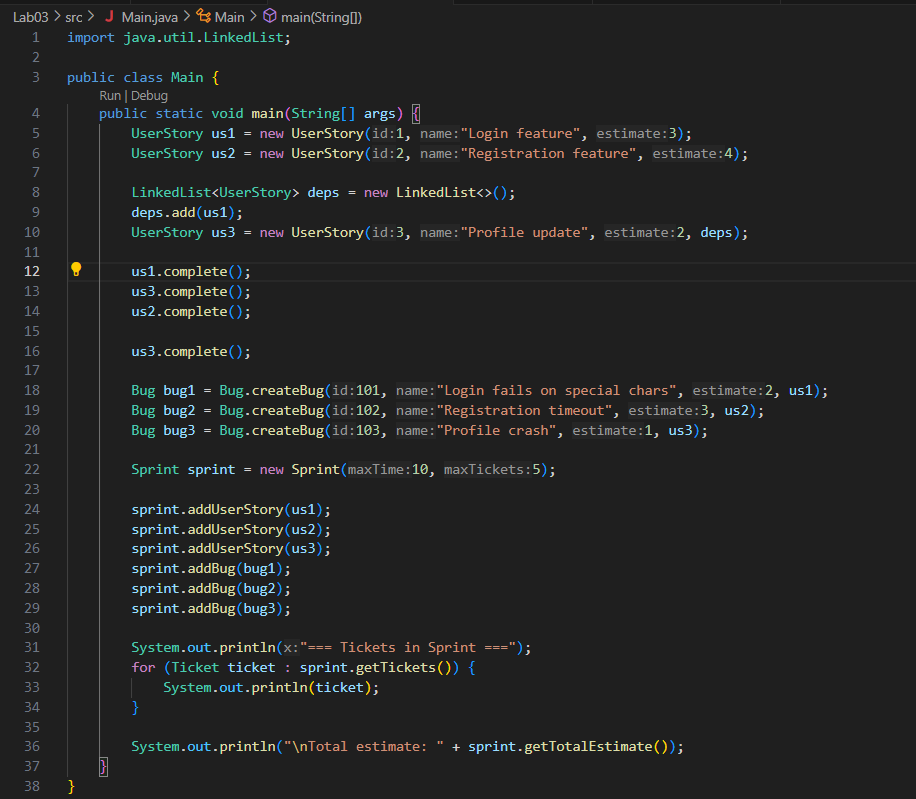
• addBug(Bug bugReport) - приймає bug, якщо він не дорівнює null і не завершений. Повертає true, якщо баг прийнято, інакше false.

• getTickets() - повертає захищену копію масиву ticket'ів на спринт. Переконайтеся, що ticket'и розташовані в тому ж порядку, як вони були прийняті в спринт.

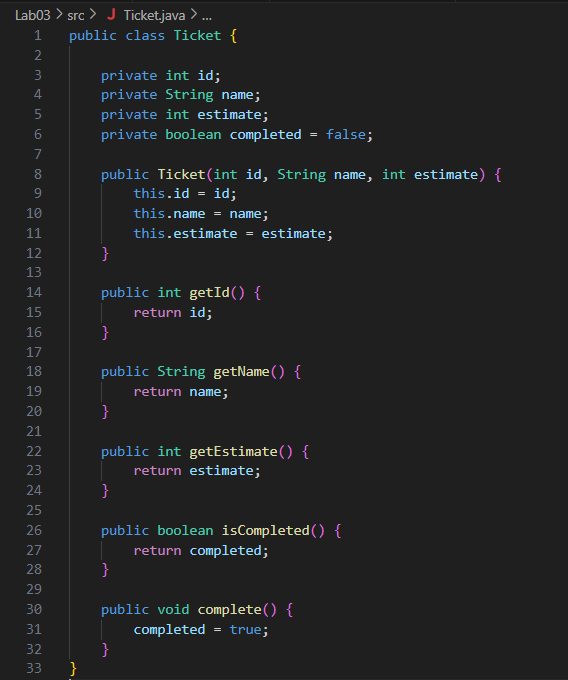
• getTotalEstimate() - повертає суму оцінок часу виконання всіх ticket'ів, прийнятих на спринт.

**Хід виконання завдання:**

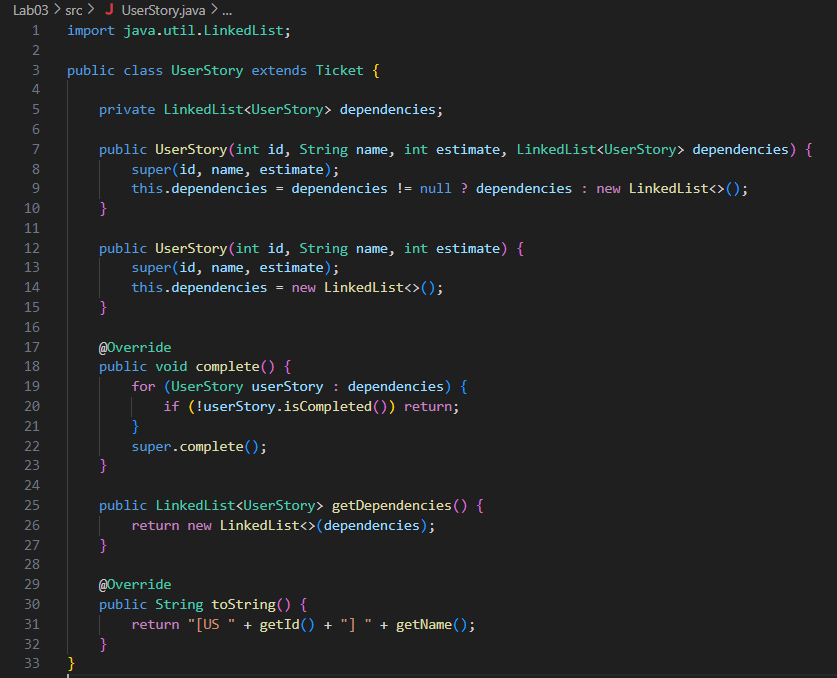
**Файл Main.java**

****

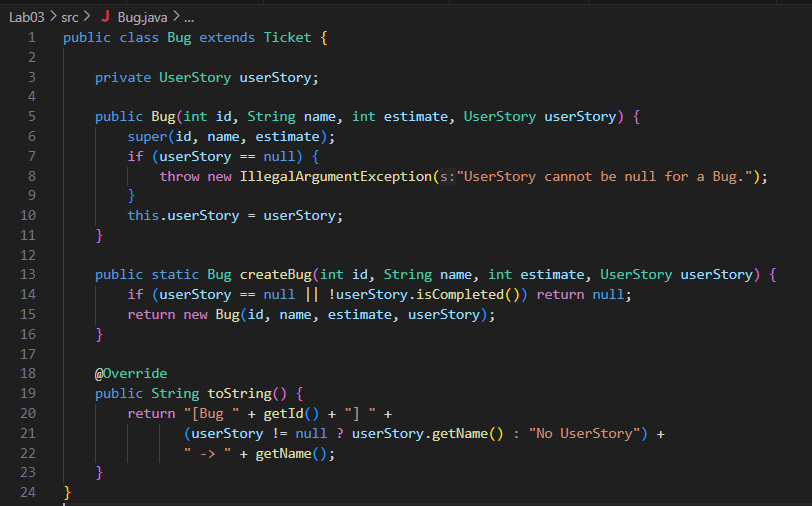
**Файл Ticket.java**

****

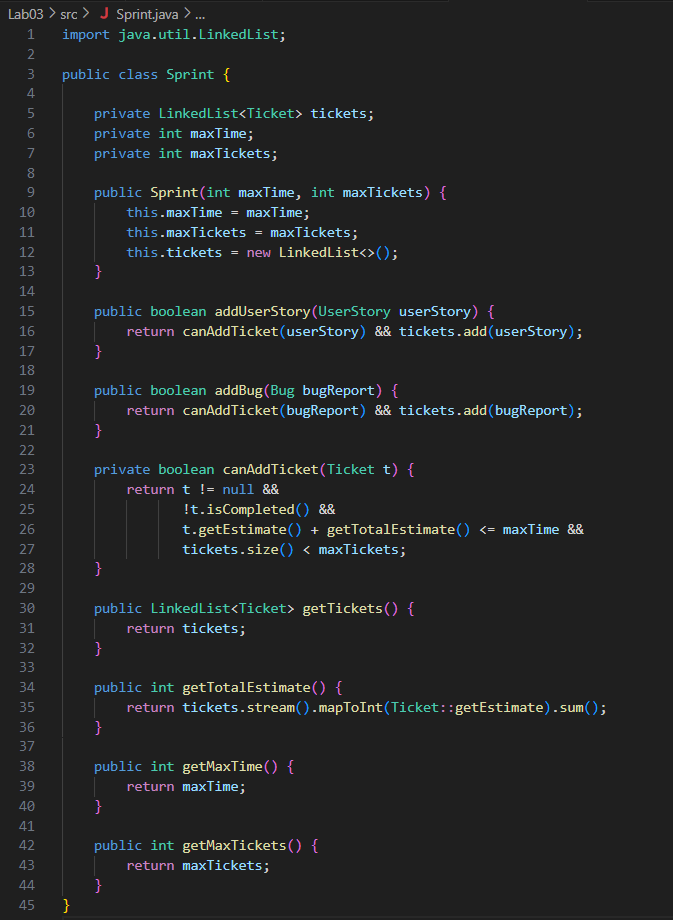
**Файл UserStory.java**

****

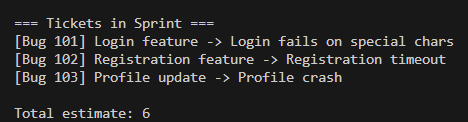
**Файл Bug.java**

****

**Файл Sprint.java**

****

**Вивід:**

****

**Текст програми:**

**Файл Main.java**

|  |
| --- |
| import java.util.LinkedList;  public class Main {  public static void main(String[] args) {  UserStory us1 = new UserStory(1, "Login feature", 3);  UserStory us2 = new UserStory(2, "Registration feature", 4);  LinkedList<UserStory> deps = new LinkedList<>();  deps.add(us1);  UserStory us3 = new UserStory(3, "Profile update", 2, deps);  us1.complete();  us3.complete();  us2.complete();  us3.complete();  Bug bug1 = Bug.createBug(101, "Login fails on special chars", 2, us1);  Bug bug2 = Bug.createBug(102, "Registration timeout", 3, us2);  Bug bug3 = Bug.createBug(103, "Profile crash", 1, us3);  Sprint sprint = new Sprint(10, 5);  sprint.addUserStory(us1);  sprint.addUserStory(us2);  sprint.addUserStory(us3);  sprint.addBug(bug1);  sprint.addBug(bug2);  sprint.addBug(bug3);  System.out.println("=== Tickets in Sprint ===");  for (Ticket ticket : sprint.getTickets()) {  System.out.println(ticket);  }  System.out.println("\nTotal estimate: " + sprint.getTotalEstimate());  }  } |

**Файл Ticket.java**

|  |
| --- |
| public class Ticket {  private int id;  private String name;  private int estimate;  private boolean completed = false;  public Ticket(int id, String name, int estimate) {  this.id = id;  this.name = name;  this.estimate = estimate;  }  public int getId() {  return id;  }  public String getName() {  return name;  }  public int getEstimate() {  return estimate;  }  public boolean isCompleted() {  return completed;  }  public void complete() {  completed = true;  }  } |

**Файл UserStory.java**

|  |
| --- |
| import java.util.LinkedList;  public class UserStory extends Ticket {  private LinkedList<UserStory> dependencies;  public UserStory(int id, String name, int estimate, LinkedList<UserStory> dependencies) {  super(id, name, estimate);  this.dependencies = dependencies != null ? dependencies : new LinkedList<>();  }  public UserStory(int id, String name, int estimate) {  super(id, name, estimate);  this.dependencies = new LinkedList<>();  }  @Override  public void complete() {  for (UserStory userStory : dependencies) {  if (!userStory.isCompleted()) return;  }  super.complete();  }  public LinkedList<UserStory> getDependencies() {  return new LinkedList<>(dependencies);  }  @Override  public String toString() {  return "[US " + getId() + "] " + getName();  }  } |

**Файл Bug.java**

|  |
| --- |
| public class Bug extends Ticket {  private UserStory userStory;  public Bug(int id, String name, int estimate, UserStory userStory) {  super(id, name, estimate);  if (userStory == null) {  throw new IllegalArgumentException("UserStory cannot be null for a Bug.");  }  this.userStory = userStory;  }  public static Bug createBug(int id, String name, int estimate, UserStory userStory) {  if (userStory == null || !userStory.isCompleted()) return null;  return new Bug(id, name, estimate, userStory);  }  @Override  public String toString() {  return "[Bug " + getId() + "] " +  (userStory != null ? userStory.getName() : "No UserStory") +  " -> " + getName();  }  } |

**Файл Sprint.java**

|  |
| --- |
| import java.util.LinkedList;  public class Sprint {  private LinkedList<Ticket> tickets;  private int maxTime;  private int maxTickets;  public Sprint(int maxTime, int maxTickets) {  this.maxTime = maxTime;  this.maxTickets = maxTickets;  this.tickets = new LinkedList<>();  }  public boolean addUserStory(UserStory userStory) {  return canAddTicket(userStory) && tickets.add(userStory);  }  public boolean addBug(Bug bugReport) {  return canAddTicket(bugReport) && tickets.add(bugReport);  }  private boolean canAddTicket(Ticket t) {  return t != null &&  !t.isCompleted() &&  t.getEstimate() + getTotalEstimate() <= maxTime &&  tickets.size() < maxTickets;  }  public LinkedList<Ticket> getTickets() {  return tickets;  }  public int getTotalEstimate() {  return tickets.stream().mapToInt(Ticket::getEstimate).sum();  }  public int getMaxTime() {  return maxTime;  }  public int getMaxTickets() {  return maxTickets;  }  } |

Посилання на github: <https://github.com/Lizarebenokk/OOP-KB-232-Liza-Rebenok/tree/main>